Contents

Chapter 1

Introduction to the Library

1.1 Introduction

The **S**ubroutine **Li**brary in **Co**ntrol and Systems **T**heory (SLICOT) is a library of widely used control system design algorithms. The intention of this library is to form the basis of new design software packages, thus avoiding duplication of software development and allowing a concentration of effort in establishing a standard set of, as far as possible, numerically robust routines with known performances in terms of reliability and efficiency.

In order to achieve the latter objective it will be necessary to take account of the latest developments in numerical analysis techniques, as applicable to control algorithms, and to consult widely with experts in the field of numerical analysis, principally numerical linear algebra.

Of no less importance however is the need to establish software documentation and implementation standards. This is necessary in order to ensure a uniform, understandable user interface, fundamental to user acceptance of the library routines and to ensure portability, reliability, and ease of maintenance of the software itself, particularly where the software may be used and/or modified in a variety of environments, by a variety of users.

This report about standards concerning the routines in SLICOT has been written in order to guide potential contributors to the library. Hopefully, it may also stimulate people to contribute.

The material used to specify the standards has been drawn from a number of sources, details of which are given in the list of references at the end of the report.

No claims are made for the completeness of the information here in defining a software standard, but rather the aim has been to provide a set of simple understandable guidelines which nevertheless go a long way towards achieving the objectives described above.

The applicability of the library is substantially enhanced when its routines are embedded in a user-friendly and widely accepted standard environment like MATLAB by an appropriate interface. Such an environment provides the user a flexible and easy way of combining and experimenting with the routines of the library and the tools of MATLAB. Moreover, the use of FORTRAN-written routines within MATLAB often has considerable advantages with respect to execution speed compared to similar genuine MATLAB m-files. Such an interface with MATLAB is described in [8].

For convenience of future contributors a summary of what a contribution to the library should consist of is given as a checklist in Appendix E.

1.2 Structure of the Library

1.2.1 The SLICOT library is divided into chapters, each devoted to a global area. Each area is denoted by a specific letter and title, e.g.,

M – Mathematical Routines
S – Synthesis Routines

Each chapter is divided into subchapters, devoted to a more specific problem area and denoted by a second letter and a title, e.g.,

MB – Linear Algebra
SB – State-space Synthesis

Each subchapter is divided into sections. A section, concerning a class of problems, is denoted by a number and a title, e.g.,

MB05 – Matrix Functions
SB02 – Riccati Equations

A subchapter may consist of only one section.

The complete list of chapters, subchapters, and sections is given in Appendix A.

1.2.2 The library will contain two categories of routines accessible to users:

Fully documented (user-callable) routines which are each documented by a Library Routine Document in the Library Manual.

Supporting (lower-level) routines which are intended to be used, not by the generality of users, but by experts, by software developers, and especially by contributors to the library. The Library Manual will only contain a list of the names of such routines and their capabilities. More detailed documentation will be provided by in-line documentation only.

The supporting routines are a privileged selection from the general collection of auxiliary routines. They give users access to separate components of the computations performed by the fully documented routines.

As a rough guide, a fully documented routine should be designed to solve a complete problem (e.g., to solve a Lyapunov equation). A supporting routine would perform some part of the necessary computations (e.g., solve a Lyapunov equation where the coefficient matrix is in real Schur form).

1.2.3 In each chapter the first subchapter is reserved for service routines (auxiliary routines) to perform elementary or subsidiary computations.

1.2.4 The policy of the library is not to introduce an unnecessary number of routines into it. In order to perform a number of related tasks, or to perform a main task with a variety of subtasks, a single routine with a mode parameter is often a satisfactory design, provided that most of the parameters are used for all option-settings.

1.2.5 It is important that parts of the library should not develop in isolation. Contributors in related areas should discuss areas of overlap or interaction. Duplication of code in different chapters must be avoided. There must be consistency and compatibility between routines in all chapters of the library.

1.3 Choice of Algorithms

The main criteria for including an algorithm in the library are:

Usefulness: An algorithm must solve problems of practical utility to at least some section of the intended users of the library. It should not be included simply because it implements an elegant numerical technique, or for purely educational reasons.

Robustness: An algorithm must either return reliable results, or it must return an error or warning indicator, if the problem has not been well-posed, or if the problem does not fall in the class to which the algorithm is applicable, or if the problem is too ill-conditioned to be solved in a particular computing environment. If this requirement is too stringent (as it may be in some chapters of the library), the documentation must make the possible risks very clear.

Numerical stability and accuracy: Algorithms should return results that are as good as can be expected when working at a given precision. If available at low cost, algorithms should provide a parameter which estimates the accuracy actually achieved. The documentation should be able to give a clear simple statement of the accuracy to be expected, if possible as a rigorous upper bound, otherwise as a conservative estimate.

Speed: An algorithm should never be chosen for its speed if it fails to meet the usual standards of robustness, numerical stability, and accuracy, as described above. Speed should be evaluated across a wide range of computing environments. Therefore it should usually be considered in terms of the number of floating-point operations, or the number and cost of iterations.

The requirements of vector-processors must be taken into account, and may imply a different choice of algorithms than those of scalar processors. If a reasonable compromise is not possible, more than one algorithm for the same problem may need to be included in the library.

1.4 Naming Convention

Every user-callable or lower-level routine in the library must have a name of the following structure:

<**chapter** > < **section** > < **free** > < **type** >

in which

chapter: < two letters > corresponding to the chapter (first letter) and the subchapter (second letter) in the Library Index in which the routine is located.

section: < two digits > corresponding to the section in the chapter in the Library Index in which the routine is located.

free: < letter > which may freely be chosen.

type: For a user-callable routine this < letter > must be:

S for a single precision routine;

D for a double precision routine;

C for a complex single precision routine;

Z for a complex double precision routine.

For a lower-level routine this < letter > may be every letter, except C, D, S, and Z.

Note that the current release of SLICOT does not contain single precision routines (real or complex).

Example: The subroutine **AB01CD** is an Analysis Routine **(A)** in Subchapter State-space Analysis **(AB)**, Section 1 **(01)** of type DOUBLE PRECISION **(D).**

1.5 User Manual

The SLICOT User Manual is organized in chapters. Each chapter consists of:

- an introduction to the problem area;
- a table of contents, with a list of all fully documented (user-callable) routines of the chapter;
- a Library Routine Document for each routine.

The manual starts with a general introduction (Introduction, Library Contents, Library Index, Keyword Index) and ends with a list of all supporting (lower-level) routines.

Chapter 2

Software Implementation Standards

2.1 General Rules

2.1.1 Subprograms must be written in standard FORTRAN77 as specified in ANSI X3.9-1978, ISO 1539-1980(E) with a number of additional restrictions (see Section 2.3). If possible, they should be verified by passing them through a FORTRAN77 verifier (e.g. [7]). The only extension to the FORTRAN77 standard is the use of a double precision complex data type (COMPLEX*16) in some routines. There is no provision for this data type in the standard.

2.1.2 Subprograms must be independent of the problem size and should be capable of being called by a program which is designed to handle problems of different sizes.

2.1.3 In general subprograms must not carry out internally any input or output of data to external devices, such as terminals, printers, discfiles, etc. Exceptions to this rule are: error routines (they should write on a specified file) and I/O routines meant for interactive usage.

2.1.4 Every effort must be made to avoid program failure within subprograms due to bad data, etc. The preferred response to failure conditions is an orderly return of control to the calling program together with the setting of an appropriate well defined error flag.

2.2 User Interface Standards

2.2.1 All input data required by the subprogram and all output data to be returned to the calling program must be transferred via the argument list. No data essential to the operation of the subprogram may be transferred via COMMON statements. Specific constants, such as characteristics of the computational environment should be provided for via an appropriate function or subroutine call within the routine.

2.2.2 The order of the parameters in the argument list in every user-callable routine must be as follows:

- mode parameters;

- user-supplied functions;
- problem dimensions;
- scalar arguments (except for leading dimensions) defining the input data (some of them may be overwritten by results);
- array arguments (directly followed by their leading first dimension and second dimension if applicable) defining the input data (some of them may be overwritten by results);
- other scalar arguments returning results;
- other array arguments returning results (directly followed by their leading first dimension and second dimension if applicable);
- arguments defining tolerances;
- work arrays (and associated array dimensions);
- warning indicator IWARN;
- error indicator INFO.

Note: input parameters may also be output parameters.

2.2.3 There should be no parameter in the argument list which is a function of other arguments.

2.2.4 Mode parameters, problem dimensions, leading dimensions, and tolerances must not be overwritten by output data.

2.2.5 Mode parameters are used to specify which option of a subprogram is required by the calling program. They should be usually of type CHARACTER*1.

The values of mode parameters should be meaningful letters, e.g., 'C' or 'c' if a routine for Lyapunov equations should solve the continuous-time equation and 'D' or 'd' if the discrete-time equation is to be solved. Capitals and the corresponding small letters must have the same meaning.

Where appropriate, mode parameters could also be of type LOGICAL or INTEGER.

2.2.6 Problem dimensions equal to zero should be allowed, where appropriate.

2.2.7 All one-dimensional arrays (unless of fixed size) must be declared either with assumed-size dimension (*) or with adjustable dimension. The assumed-size dimension (*) must be used if the actual dimension is allowed to be zero (empty actual array) or if the dimension is not a simple integer expression in terms of parameters in the argument list. Similar remarks also apply to the last dimension of two-dimensional or three-dimensional arrays (see below).

Each two-dimensional array must have its own leading dimension, to be passed as a separate integer parameter, directly following the array-name in the argument list. For this integer parameter a name of the form LD<array-name> must be used.

For example, the two-dimensional array A, which is supposed to contain an n-by-n matrix ($n \geq 1$), is declared as:

```
SUBROUTINE        XXXXXX (N, A, LDA, ...)
INTEGER           N, LDA
DOUBLE PRECISION  A(LDA, N)
```

Each three-dimensional array must have its own leading dimensions, to be passed as separate integer parameters, directly following the array-name in the argument list. For these integer parameters names of the form LD<array-name>1 and LD<array-name>2 must be used.

E.g., the three-dimensional array B of which the last dimension is at least max(M,N) is declared as:

```
SUBROUTINE        XXXXXX (N, M, B, LDB1, LDB2, ...)
INTEGER           N, M, LDB1, LDB2
DOUBLE PRECISION  B(LDB1, LDB2, *)
```

2.2.8 Arrays with more than three dimensions must not be used and three-dimensional arrays should be avoided wherever possible.

2.2.9 Tolerances are used to control numerical error. If more than one tolerance is required, then the use of the names TOL1, TOL2, TOL3, ... is recommended. Otherwise, the name TOL should be used. Where appropriate, other names could be used, e.g., RCOND for the minimal value of the reciprocal condition number, below which a matrix is considered to be singular.

All tolerances must be reset internally by the routine if the accuracy requirements specified by the user cannot be achieved, e.g., if TOL $<$ 0.0 on entry, then the tolerance might be taken as EPS (machine precision) instead.

2.2.10 Only one work array of each type may occur in the argument list of every user-callable routine, where the following names must be used:

IWORK for integer workspace;

SWORK for single precision workspace;

DWORK for double precision workspace;

CWORK for complex single precision workspace;

ZWORK for complex double precision workspace;

BWORK for logical workspace.

Work arrays must appear in the above order in the argument list of the routine.

If several work arrays of the same type are required in the body of a subprogram, then the single work array may be partitioned via a call to a lower-level routine.

Workspace is provided in either of the following forms:

Fixed size workspace, which is applied when the size of the workspace is a simple function of the problem dimensions and the mode parameters.

Variable size workspace, which is applied when the size of the workspace is a complicated function of the problem dimensions and the mode parameters or when the performance of the routine can be improved by increasing the size of workspace. Mostly, the latter is the case when the routine is an implementation of a block algorithm or when it calls a blocked routine.

In the argument list the work array xWORK (where x is one of the capitals I, S, D, C, Z, B) must be followed by an input integer parameter LxWORK, which contains the size of the workspace available. In case of a blocked routine the optimal value of LxWORK is a function of the problem dimensions, the mode parameters, and the optimal block size NB, returned by the LAPACK routine ILAENV. For further information, see [1] Sections 5.2 and 6.2.

On exit, if possible, xWORK(1) should contain the optimal size of workspace.

See also 3.2.1 for examples.

2.2.11 The optional warning indicator IWARN is an integer variable used by the routine to indicate that either the normal situation (IWARN $= 0$) or an exceptional situation (IWARN $\neq 0$) occurs. Unless the routine generates a warning, IWARN must contain 0 on exit. Other values of IWARN may be used to indicate that a non-fatal "error" has occurred (that is, one which does not effect an immediate return to the calling program), e.g., lowering the rank of a matrix. Note that any results produced by the routine must still be correct even if IWARN $\neq 0$ on exit.

2.2.12 The error indicator INFO must be included as the final parameter in the argument list of every user-callable routine. Unless the routine detects an error, INFO must contain 0 on exit. Other values of INFO may be used to indicate that a fatal error has occurred (that is, one which effects an immediate return to the calling program), e.g., violation of the conditions of an input parameter or attempted division by zero.

If a subroutine detects an error in the i-th parameter of the argument list before the actual computation starts, it must return INFO $= -i$. Otherwise, if an error is detected in the course of the computation, INFO must contain a positive value (see also 3.2.1).

2.2.13 All input data must be unaltered on return from the subprogram unless this is unlogical due to the nature of the parameter or impractical due to excessive storage requirements. A possible change of the contents of input parameters must be emphasized in the documentation and in-line comments of the subprogram.

2.3 Programming Standards

2.3.1 Subprograms should be well structured. The structure should essentially consist of a subdivision into linearly ordered modules, where each module itself may consist of a set of submodules. Wherever possible, use should be made of modules from accepted infrastructural packages (e.g., BLAS or LAPACK) or of modules already included in the Library.

2.3.2 Comment statements must be inserted where needed to clarify the execution and data flow in the program. Comments must be in English and written in normal printed text format (not all upper case).

2.3.3 Every subprogram (user-callable and lower-level) must contain the WGS copyright note and an in-line documentation in its heading. The in-line documentation consists of the following sections:

- Purpose;
- Arguments;
- Method;
- References;
- Contributors;
- Revisions;
- Keywords.

For a detailed description of the first four points see 3.2.1. See also Appendix B for an example.

2.3.4 The following restrictions, additional to those imposed by the FORTRAN77 standard, are regarded as mandatory by WGS:

- Do not use COMMON, except for communication between library routines. In any case avoid the use of COMMON if possible.
- Do not use EQUIVALENCE.
- Do not use ASSIGN, assigned GO TO, computed GO TO.
- Do not use BLOCK DATA.
- Do not use REAL or DOUBLE PRECISION variables to control DO-loops.
- Do not use ENTRY and alternate RETURN.
- Do not use FORTRAN keywords or intrinsic functions as names of variables or subprograms.

2.3.5 Meaningful variable names should be used. Where a routine is based on referenced published material, variable names should match as closely as possible the nomenclature of the published material. Ensure that variables and subprograms have distinct names.

2.3.6 All arguments must be explicitly typed, i.e., appear in a type statement.

2.3.7 All local variables, i.e., those variables in the body of the routine which are not in the argument list, should be specified in type statements within the routine separately from the arguments.

For development purpose the use of the (non-standard) statement IMPLICIT NONE, if available, is recommended on top of the specification of the local variables.

2.3.8 To give a name to values which are truly constant, PARAMETER statements rather than DATA statements should be used, e.g.,

```
DOUBLE PRECISION  ZERO, ONE
PARAMETER         (ZERO=0.0D0, ONE=1.0D0)
```

DATA statements can be used to assign values to array-elements, or to assign initial values to scalar variables whose values may subsequently be changed.

2.3.9 REAL or DOUBLE PRECISION constants should be defined with at least 20 significant digits (and with no leading zeros) unless their values can be exactly represented in fewer digits (e.g., 1.5) or it is not necessary for the values to be accurate to full machine precision. Rational values should be expressed as rational fractions (e.g., one-third as 1.0/3.0).

2.3.10 Before starting the actual computation the subroutine must check the input parameters for errors and zero dimensions.

The following types of errors in input parameters should be taken into account:

- negative problem dimensions;
- illegal values of mode parameters;
- non-positive leading dimensions;
- insufficient size of leading dimensions with respect to the problem dimensions;
- insufficient value of workspace size parameter LxWORK (where x is one of the capitals I, S, D, C, Z, L) in case of variable size workspace (see 2.2.10);
- other errors in input parameters arising from the context, e.g., an impossible combination of the values of mode parameters.

If zero dimensions are encountered, which lead to immediate termination, the subroutine must set INFO = 0 and return to the calling routine.

2.3.11 DO statements must have an associated CONTINUE statement to indicate the extent of the resultant loop, and the body of the statements in the loop should be indented.

2.3.12 DO statements should be used to execute a repetition for a given number of times, i.e., the index variable should pass through all values of the list and no GO TO statement should be used to jump out of the DO loop.

Repetitions with an unforeseen number of executions should be implemented by a WHILE construction e.g.,

```
      I = 1
C     WHILE (I <= MAXI and A(I) > 0.0) DO
   10 IF (I .LE. MAXI) THEN
         IF (A(I) .GT. ZERO) THEN
            ...
            I = I + 1
            GO TO 10
         END IF
      END IF
C     END WHILE 10
```

2.3.13 The use of GO TO statements should be avoided as far as possible. Arithmetic IF statements should not be used at all. The use of IF THEN, ELSE, and ELSE IF statements is recommended instead. The body of these statements should be indented.

2.3.14 Statement labels must appear in ascending order right adjusted in the label field.

2.3.15 Computations on arrays should be programmed 'by columns' wherever possible, in order to reduce the amount of paging in virtual-memory systems, and to increase efficiency on some vector-processing systems. Also to facilitate automatic vectorization, innermost DO loops should be kept simple and, if possible, should not contain IF statements (this is especially important in computationally intensive parts of the code).

2.3.16 Routines should be coded so that the algorithms adapt to characteristics of the computational environment in which they are being executed. Relevant characteristics (e.g., machine base, relative machine precision, largest/smallest machine representable number, safe range parameter) are provided by the LAPACK routine DLAMCH.

2.3.17 The use of level 2 or 3 BLAS is strongly recommended because of efficiency and portability reasons.

2.3.18 It is recommended that the penultimate line of the source code contains a comment statement of the form:

```
C *** Last line of < subprogram name > ***
```

Chapter 3

Documentation Standards

3.1 General Rules

3.1.1 Every user-callable subprogram must be documented by a Library Routine Document.

3.1.2 Documentation must provide complete information to enable a user of the subprogram to proceed with a high expectation of successful application.

In particular, limitations or constraints on the use of a subprogram should be emphasized and if possible advice given for overcoming the limitations by the use of alternative subprograms, choice of mode of use, etc..

3.1.3 For every user-callable routine, an example program demonstrating some of the capabilities of the routine must be provided. Note that example programs, together with their data files (if any) and results files, are reproduced in their entirety in the manual. Neither the example program, nor the input data nor the results should be any longer than is absolutely necessary.

3.2 Subprogram Documentation Standards

3.2.1 The Library Routine Document for a user-callable routine must consist of the following numbered sections:

1 Purpose
This section must contain a brief description outlining the purpose of the routine.

2 Specification
This section must contain the heading of the subprogram with the list of all the parameters in the argument list along with a type declaration for each parameter, e.g.,

```
      SUBROUTINE AB01OD( STAGES, JOBU, JOBV, N, M, A, LDA, B, LDB, U,
     $                   LDU, V, LDV, NCONT, INDCON, KSTAIR, TOL, IWORK,
     $                   DWORK, LDWORK, INFO )
C     .. Scalar Arguments ..
      CHARACTER*1       STAGES, JOBU, JOBV
```

```
      INTEGER           N, M, LDA, LDB, LDU, LDV, NCONT, INDCON, LDWORK,
     $                  INFO
      DOUBLE PRECISION  TOL
C     .. Array Arguments ..
      INTEGER           IWORK(*), KSTAIR(*)
      DOUBLE PRECISION  A(LDA,*), B(LDB,*), DWORK(*), U(LDU,*), V(LDV,*)
```

Note that assumed size (∗) dimensions are not to be used in this section if it is possible to use an integer expression instead. Consistently, '$' should be used as the line continuation character.

3 Arguments
This section must contain a list of all parameters along with their description. It is subdivided into the Subsections:

- Mode Parameters;
- Input/Output Parameters;
- Tolerances;
- Workspace;
- Warning Indicator;
- Error Indicator.

The order of the entries in this section must accord with the order of the parameters in the calling sequence of the subroutine (see 2.2.2).

Each entry must have the form:

$<$ **name** $>$ $<$ **I/O-note** $>$ $<$ **type** $>$ $<$ **array-dimensions** $>$
$<$ **description** $>$ $<$ **restrictions** $>$

where the synonyms have the following meaning:

name: The name of the parameter.

I/O-note: If the parameter is an input/output parameter (except for leading dimensions) it must contain the phrase:

- **(input)** in case of an input parameter, the contents of which is not overwritten on exit;
- **(output)** in case of an output parameter, the input contents of which does not affect any output data of the subroutine;
- **(input/output)** in case of an input parameter, the contents of which may be overwritten on exit, possibly by useless data.

type: The type of the parameter.

array-dimensions: In case of an array argument the array dimensions.

description: A concise description of the parameter.

restrictions: Restrictions imposed on the parameter (if there are any).

In the sequel, some comments and examples for each subsection are provided.

3.1 Mode Parameters
Example:

STAGES	CHARACTER*1 Specifies the reduction stages to be performed as follows: = 'F': Perform the forward stage only; = 'B': Perform the backward stage only; = 'A': Perform both (all) stages.

3.2 Input/Output Parameters
Example:

A	(input/output) DOUBLE PRECISION array, dimension (LDA,N) On entry, the leading n-by-n part of this array must contain the state transition matrix A to be transformed. ... On exit, the leading n-by-n part of this array contains the transformed state transition matrix $U^T AU$. ...
LDA	INTEGER The leading dimension of array A. LDA $\geq$ max(1,N).

It must be mentioned explicitly in the description if an output array argument contains no useful information on exit, e.g.,

> On exit, this array contains no useful information.

3.3 Tolerances
The description of a tolerance parameter must contain an outline of how the default tolerance may be invoked (if applicable), e.g.,

TOL	DOUBLE PRECISION The tolerance to be used in rank determination when transforming (A, B). If the user sets TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number ...

3.4 Workspace
Workspace is provided either as fixed size or variable size workspace (see 2.2.10). In case of variable size workspace the description should contain an expression for the optimal size of workspace or a note on how to get the optimal or at least a good value for LxWORK (where x is one of the capitals I, S, D, C, Z, B)

Example for fixed size workspace:

DWORK	(workspace) DOUBLE PRECISION array, dimension (N)

Example for variable size workspace:

DWORK	DOUBLE PRECISION array, dimension (LDWORK) On exit, if INFO = 0, DWORK(1) returns the optimal value of LDWORK.

LDWORK	INTEGER The length of the array DWORK. If STAGES $\neq$ 'B', LDWORK $\geq$ max(1, N$*$M + max(N,M) + max(N,3$*$M)); if STAGES = 'B', LDWORK $\geq$ max(1, M + max(N,M)). For optimum performance LDWORK should be larger.

Note that assumed size ($*$) dimensions are not to be used.

3.5 Warning Indicator

Example:

IWARN	INTEGER	
	=0:	No warning;
	=1:	The matrix A is rank deficient.

3.6 Error Indicator

The description of the diagnostic argument must have the form:

INFO	INTEGER	
	=0:	successful exit;
	<0:	if INFO = $-i$, the i^{th} argument had an illegal value;
	=1:	. . . ;
		$\vdots$
	=n:	. . . .

where the lines 1 to 3 are mandatory.

4 Method

This section must contain a description of the algorithm used by the routine.

5 References

This section must contain a list of references which the user may consult for further information. Note that theses and internal reports should not be referenced if a reference in the open literature is available.

6 Numerical Aspects

In this section information about the algorithm concerning accuracy, numerical stability, and operations count (if applicable) can be given.

7 Further Comments

This section is meant for additional remarks about the routine which may be of interest to users, e.g.,

> This routine may be BASE (machine base) dependant.

8 Example

This section must contain a brief description of the example problem to be solved. See Appendix C for an example.

8.1 Program Text

A simple program to show how to use the subprogram.

8.2 Program Data

Input data of a small well conditioned problem, for which the results are reproducible on different machines.

8.3 Program Results
Output data produced by the test program with the given input data.

See Appendix C for a complete example.

3.2.2 In the Library Routine Document empty (sub)sections must be included in the text with the description 'None.'.

3.2.3 In the Library Routine Document mathematical variables must be designated by slanted letters and FORTRAN variables by Roman letters, e.g.,

The array A contains the matrix A.

3.2.4 In the Library Routine Document matrix dimensions must be written in the form $<$ dimension-1 $>$-by-$<$ dimension-2 $>$, e.g.,

A is an m-by-n matrix...

where m and n correspond to the FORTRAN variables M and N, respectively.

3.2.5 The end of a Library Routine Document must be marked by a horizontal line.

3.2.6 Every subprogram (user-callable and lower-level) must have in-line documentation inserted in the source code between the heading and the first executable statement. This documentation must contain the items which are listed in 2.3.3. See 3.2.1 for a detailed description and Appendix B for a complete example.

3.2.7 In in-line documentations the character ' should be used to denote the (conjugate) transposed matrix, e.g., `... the matrix U' * A * U ... .`

Chapter 4

Example Programs and Certification Programs

4.1 Example Programs

4.1.1 The purpose of an example program as part of the documentation of a routine, is to give users (especially inexperienced ones) an example of the straightforward application of the routine to solve a simple problem.

4.1.2 Example programs must be capable of producing similar results when run across a wide range of computing environments and should be written with a reasonable degree of generality so that they can easily be adapted to solve other problems.

4.1.3 PARAMETER statements should be used to define the input and output devices and the dimension of arrays.

4.1.4 All variables must be explicitly typed, i.e. appear in a type statement.

4.1.5 I/O statements other than formatted READ and WRITE should not be used. If possible record lengths should be kept to not more than 80 characters, to avoid line wrap-around on terminals.

See Appendix C, Section 8 for an example.

4.2 Certification Programs

4.2.1 With every routine proposed to be included in the library a certification program must be supplied. The purpose of a certification program is to validate a new routine or subsequent modifications, and to test the implementation of the library in different computing environments.

4.2.2 The essential elements for contributors to contribute toward a certification program are:

- a small number (as small as possible) of simple (but non-trivial) test problems, which are expected to exercise all the normal paths in the code;
- additional problems (again as simple as possible), which should test extreme conditions (e.g. ill-conditioning) over a wide range of environments.

Chapter 5

Standard Data Structures

Many of the library subprograms deal with vectors and matrices. Also polynomials and vector/matrix polynomials are used in library subprograms. The purpose of this chapter is to give an introduction to the data-structures used in connection with these entities.

5.1 Vector Storage

A vector refers to a FORTRAN one-dimensional array. For example, an n-vector v is stored in an array V, which is declared as

```
DOUBLE PRECISION  V(NMAX)
```

where $n \leq$ NMAX. The vector element v_i $(1 \leq i \leq n)$ is stored in the array element V(i).

5.2 Matrix Storage

SLICOT allows the following different storage schemes for matrices:

- conventional storage in a two-dimensional array;
- packed storage for symmetric, Hermitian or triangular matrices;
- band storage for band matrices;
- the use of two or three one-dimensional arrays to store bidiagonal or tridiagonal matrices.

These storage schemes are compatible with those used in LAPACK [1] and in the BLAS [3, 4, 6].

In the examples below, '$*$' indicates an array element that need not be set and is not referenced by SLICOT routines. Elements that "need not be set" are never read, written to, or otherwise accessed by the SLICOT routines. The examples illustrate only the relevant part of the arrays. Array arguments may of course have additional rows or columns, according to the usual rules for passing array arguments in FORTRAN77.

5.2.1 Conventional Storage

General Matrices

In general, a matrix is stored in an array with two dimensions. For example, an m-by-n matrix $A = [a_{ij}]$ can be stored in an array A, which is declared as

```
DOUBLE PRECISION  A(LDA,NMAX)
```

where $m \leq$ LDA and $n \leq$ NMAX. The matrix element a_{ij} $(1 \leq i \leq m,\ 1 \leq j \leq n)$ is contained in the array element A(i,j).

Frequently, the dimensions of the matrix and the associated array do not accord, i.e. the matrix does not fill the whole array. The array elements are arranged in the memory in the following manner (for example: LDA = 3, NMAX = 4, $m = 2$, $n = 3$):

$$a_{11}\ a_{21}\ *\ |\ a_{12}\ a_{22}\ *\ |\ a_{13}\ a_{23}\ *\ |\ *\ \ *\ \ *$$

Note that FORTRAN77 stores two-dimensional arrays by columns.

As a consequence, if a matrix is used in a subprogram not only the actual dimensions have to be given but also the leading dimension (in the above example: LDA), e.g.

```
CALL SUBR (M, N, A, LDA, ... )
```

See also 2.2.7.

Note that submatrices can easily be accessed by subprograms, e.g.

```
CALL SUBR (M2, N2, A(M1,N1), LDA, ... )
```

where the subroutine SUBR refers to the M2-by-N2 submatrix $A(\mathrm{M1} : \mathrm{M1} + \mathrm{M2} - 1, \mathrm{N1} : \mathrm{N1} + \mathrm{N2} - 1)$.

Triangular and Hessenberg Matrices, Matrices in Real Schur Form

If a matrix is triangular (upper or lower, as specified by the argument UPLO), only the elements of the relevant triangle are accessed. The remaining elements of the array need not be set. Example for a triangular matrix when $n = 4$:

UPLO	Triangular matrix A	Storage in array A
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{matrix}$
'L'	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$

Similarly, if the matrix is in upper Hessenberg or upper real Schur form, elements below the first subdiagonal need not be set.

Symmetric and Hermitian Matrices

Routines that handle symmetric or Hermitian matrices allow for either the upper or lower triangle of the matrix (as specified by UPLO) to be stored in the corresponding elements of the array. The remaining elements of the array need not be set. For example, when $n = 4$:

UPLO	Hermitian matrix A	Storage in array A
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} \\ \bar{a}_{14} & \bar{a}_{24} & \bar{a}_{34} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{matrix}$
'L'	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & \bar{a}_{41} \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$

5.2.2 Packed Storage

Symmetric, Hermitian or triangular matrices may be stored more compactly, if the relevant triangle (again as specified by UPLO) is packed *by columns* in a one-dimensional array. In SLICOT, arrays that hold matrices in packed storage, have names ending in 'P'. So:

- if UPLO = 'U', a_{ij} is stored in AP$(i + j(j-1)/2)$ for $i \le j$;
- if UPLO = 'L', a_{ij} is stored in AP$(i + (2n-j)(j-1)/2)$ for $i \ge j$.

For example:

UPLO	Triangular matrix A	Packed storage in array AP
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$a_{11} \mid a_{12}\ a_{22} \mid a_{13}\ a_{23}\ a_{33} \mid a_{14}\ a_{24}\ a_{34}\ a_{44}$
'L'	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$a_{11}\ a_{21}\ a_{31}\ a_{41} \mid a_{22}\ a_{32}\ a_{42} \mid a_{33}\ a_{43} \mid a_{44}$

UPLO	Hermitian matrix A	Packed storage in array AP
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} \\ \bar{a}_{14} & \bar{a}_{24} & \bar{a}_{34} & a_{44} \end{pmatrix}$	$a_{11} \mid a_{12}\ a_{22} \mid a_{13}\ a_{23}\ a_{33} \mid a_{14}\ a_{24}\ a_{34}\ a_{44}$
'L'	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & \bar{a}_{41} \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$a_{11}\ a_{21}\ a_{31}\ a_{41} \mid a_{22}\ a_{32}\ a_{42} \mid a_{33}\ a_{43} \mid a_{44}$

Note that for real or complex symmetric matrices, packing the upper triangle by columns is equivalent to packing the lower triangle by rows; packing the lower triangle by columns is equivalent to packing the upper triangle by rows. For complex Hermitian matrices, packing the upper triangle by columns is equivalent to packing the conjugate of the lower triangle by rows; packing the lower triangle by columns is equivalent to packing the conjugate of the upper triangle by rows.

5.2.3 Band Storage

General Band Matrices

An m-by-n band matrix with k_l subdiagonals and k_u superdiagonals may be stored compactly in a two-dimensional array with $k_l + k_u + 1$ rows and n columns. Columns of the matrix are stored in corresponding columns of the array, and diagonals of the matrix are stored in rows of the array. This storage scheme should be used in practice only if $k_l, k_u \ll \min(m, n)$, although SLICOT routines work correctly for all values of k_l and k_u. In SLICOT, arrays that hold matrices in band storage have names ending in 'B'.

To be precise, a_{ij} is stored in $\mathrm{AB}(k_u + 1 + i - j, j)$ for $\max(1, j - k_u) \leq i \leq \min(m, j + k_l)$. For example, when $m = n = 5$, $k_l = 2$ and $k_u = 1$:

Band matrix A	Band storage in array AB
$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & a_{43} & a_{44} & a_{45} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

The elements marked $*$ in the upper left and lower right corners of the array AB need not be set, and are not referenced by SLICOT routines. When a band matrix is supplied for an LU factorization, space must be allowed to store an additional k_l superdiagonals, generated by fill-in as a result of row interchanges. This means that the matrix is stored according to the above scheme, but with $k_l + k_u$ superdiagonals.

Triangular Band Matrices

Triangular band matrices are stored in the same format, with either $k_l = 0$ if upper triangular, or $k_u = 0$ if lower triangular.

Symmetric and Hermitian Band Matrices

For symmetric or Hermitian band matrices with k_d subdiagonals or superdiagonals, only the upper or lower triangle (as specified by UPLO) need be stored:

- if UPLO = 'U', a_{ij} is stored in $\mathrm{AB}(k_d + 1 + i - j, j)$ for $\max(1, j - k_d) \leq i \leq j$;

- if UPLO = 'L', a_{ij} is stored in $\mathrm{AB}(1 + i - j, j)$ for $j \leq i \leq \min(n, j + k_d)$.

For example, when $n = 5$ and $k_d = 2$:

UPLO	Hermitian band matrix A	Band storage in array AB
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & & \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} & \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} & a_{35} \\ & \bar{a}_{24} & \bar{a}_{34} & a_{44} & a_{45} \\ & & \bar{a}_{35} & \bar{a}_{45} & a_{55} \end{pmatrix}$	$\begin{matrix} * & * & a_{13} & a_{24} & a_{35} \\ * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \end{matrix}$
'L'	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & & \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} & \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} & \bar{a}_{53} \\ & a_{42} & a_{43} & a_{44} & \bar{a}_{54} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

5.2.4 Tridiagonal and Bidiagonal Matrices

An unsymmetric tridiagonal matrix of order n is stored in three one-dimensional arrays, one of length n containing the diagonal elements, and two of length $n-1$ containing the subdiagonal and superdiagonal elements in elements $1 : n-1$.

A symmetric tridiagonal or bidiagonal matrix is stored in two one-dimensional arrays, one of length n containing the diagonal elements, and one of length $n-1$ containing the off-diagonal elements.

5.2.5 Unit Triangular Matrices

SLICOT routines may have an option to handle unit triangular matrices (that is, triangular matrices with diagonal elements = 1). This option is specified by an argument DIAG. If DIAG = 'U' (Unit triangular), the diagonal elements of the matrix need not be stored, and the corresponding array elements are not referenced by the SLICOT routines. The storage scheme for the rest of the matrix (whether conventional, packed or band) remains unchanged, as described in Subsections 5.2.1, 5.2.2, and 5.2.3.

5.2.6 Real Diagonal Elements of Complex Matrices

Complex Hermitian matrices have diagonal elements that are by definition purely real. In addition, some complex triangular matrices computed by SLICOT routines are defined by the algorithm to have real diagonal elements (in Cholesky or QR factorization, for example).

If such matrices are supplied as input to SLICOT routines, the imaginary parts of the diagonal elements are not referenced, but are assumed to be zero. If such matrices are returned as output by SLICOT routines, the computed imaginary parts are explicitly set to zero.

5.2.7 Representation of Orthogonal or Unitary Matrices

In accordance with [1] a real orthogonal or complex unitary matrix (usually denoted Q) is sometimes represented in SLICOT as a product of elementary reflectors - also referred to as *elementary Householder matrices* (usually denoted H_i). For example,

$$Q = H_1 H_2 \dots H_k$$

Most users need not be aware of the details, because LAPACK routines are provided to work with this representation:

- routines whose names begin with DORG- (real) or ZUNG- (complex) can generate all or part of Q explicitly;
- routines whose names begin with DORM- (real) or ZUNM- (complex) can multiply a given matrix by Q or Q^H without forming Q explicitly.

The following further details may occasionally be useful.

An elementary Householder matrix H of order n is a unitary matrix of the form

$$H = I - \tau v v^H \tag{5.1}$$

where τ is a scalar, and v is an n-vector, with $|\tau|^2 \|v\|_2^2 = 2\mathrm{Re}(\tau)$. v is often referred to as the *Householder vector*. Often v has several leading or trailing zero elements, but for the purpose of this discussion assume that H has no such special structure.

There is some redundancy in the representation (5.1), which can be removed in various ways. The representation used in SLICOT sets $v_1 = 1$. Hence v_1 need not be stored. In real arithmetic, $1 \leq \tau \leq 2$, except that $\tau = 0$ implies $H = I$.

In complex arithmetic, τ may be complex, and satisfies $1 \leq \mathrm{Re}(\tau) \leq 2$ and $|\tau - 1| \leq 1$. Thus a complex H is not Hermitian (as it is in other representations), but it is unitary, which is the important property. The advantage of allowing τ to be complex is that, given an arbitrary complex vector x, H can be computed so that

$$H^H x = \beta(1, 0, \dots, 0)^T$$

with *real* β. This is useful, for example, when reducing a complex Hermitian matrix to real symmetric tridiagonal form, or a complex rectangular matrix to real bidiagonal form.

5.3 Polynomial Storage

SLICOT allows the storage of scalar, vector, and matrix polynomials in arrays of dimension 1, 2, or 3, respectively. Rational functions should be stored in two arrays, where the first array contains the coefficients of the numerator and the second contains the coefficients of the denominator.

5.3.1 Scalar Polynomial

The coefficients p_i of a polynomial p of degree n:

$$p(z) = p_0 + p_1 z + p_2 z^2 + ... + p_n z^n$$

are stored in a one-dimensional array P(M) where P(i+1) contains p_i and M is at least $n+1$.

5.3.2 Vector Polynomial

A vector polynomial p of degree n of which the coefficients p_i are vectors of length k, is stored in a two-dimensional array P(K,M) with $k \leq$ K and $n+1 \leq$ M. The vectors are stored in the columns of P. Consequently, the second index refers to the coefficients of the polynomial, i.e.

$$\mathrm{P}(1, i+1), \ldots, \mathrm{P}(k, i+1)$$

specifies the coefficient p_i of z^i. p may also be considered as a polynomial vector q of which the h-th element which is a polynomial of degree n is given by P(h,1),...,P(h,$n+1$). Optionally, a one-dimensional companion array, say DEGP(K), may be given of which the h-th element specifies the actual degree of the h-th polynomial element of the vector q. If this array is not given, all polynomial elements of the vector q are assumed to have the same degree.

5.3.3 Matrix Polynomial

A matrix polynomial P of degree n of which the coefficients P_i are k-by-l matrices, is stored in a three-dimensional array P(K,L,M) with $k \leq$ K, $l \leq$ L and $n+1 \leq$ M. The first two indices refer to a matrix element, the third index refers to the coefficients of the polynomial, i.e.,

$$\begin{array}{ccc} \mathrm{P}(1,1,i+1) & \ldots & \mathrm{P}(1,l,i+1) \\ \vdots & & \vdots \\ \mathrm{P}(k,1,i+1) & \ldots & \mathrm{P}(k,l,i+1) \end{array}$$

give the matrix coefficient P_i of z^i.
P may also be considered as a polynomial matrix Q of which the (h,j)-th element which is a polynomial of degree n is given by P(h,j,1),...,P(h,j,$n+1$). Optionally, a two-dimensional companion array, say DEGP(K,L), may be given of which the (h,j)-th element specifies the actual degree of the (h,j)-th polynomial element of the matrix Q. If this array is not given, all polynomial elements of the matrix Q are assumed to have the same degree.

Note that if a two-dimensional or three-dimensional array is used in a subprogram not only the actual dimensions have to be given but also all but the last of the maximum dimensions (leading dimensions).

Appendix A

Library Index

Chapter - **U: Utility Routines**

UA: Auxiliary Routines

UB: File Handling

UC: Graphics Handling

UD: Numerical Data Handling

UE: Machine Dependent Routines

UF: Text Handling

Chapter - **M: Mathematical Routines**

MA: Auxiliary Routines
- **MA01:** Mathematical Scalar Routines
- **MA02:** Mathematical Vector/Matrix Routines
- **MA03:** Sorting Routines
- **MA04:** Statistical Routines

MB: Linear Algebra
- **MB01:** Basic Linear Algebra Manipulations
- **MB02:** Linear Equations and Least Squares
- **MB03:** Eigenvalues and Eigenvectors
- **MB04:** Decompositions and Transformations
- **MB05:** Matrix Functions

MC: Polynomial and Rational Function Manipulations
- **MC01:** Scalar Polynomials
- **MC02:** Scalar Rational Functions
- **MC03:** Polynomial Matrices

MD: Optimization
- **MD01:** Basic Optimization Routines
- **MD02:** Unconstrained Linear Least Squares
- **MD03:** Unconstrained Nonlinear Least Squares
- **MD04:** Minimax Problems

TG: Generalized State-space
- **TG01:** Generalized state-space transformations (including minimal realization)
- **TG02:** Discretization
- **TG03:** Generalized state-space to polynomial matrix conversion
- **TG04:** Generalized state-space to rational matrix conversion
- **TG05:** Generalized state-space to frequency response
- **TG06:** Generalized state-space to time response
- **TG07:** Generalized state-space to state-space

Chapter - **A: Analysis Routines**

AA: Auxiliary Routines

AB: State-space Analysis (including Periodic Systems)
- **AB01:** Canonical and Quasi Canonical Forms
- **AB02:** Change of Basis
- **AB03:** Structural Indices
- **AB04:** Continuous/Discrete Time
- **AB05:** Interconnection of Subsystems
- **AB06:** Controllability, Observability
- **AB07:** Inverse and Dual Systems
- **AB08:** Poles, Zeros, Gain
- **AB09:** Model Reduction
- **AB10:** Subspace Computation
- **AB11:** Scalar and Multivariable Root Loci
- **AB12:** Control Diagrams
- **AB13:** Norms and Distances

AC: Polynomial Matrix Analysis
- **AC01:** Canonical and Quasi Canonical Forms
- **AC02:** Equivalence Transformations
- **AC03:** Greatest Common Divisor
- **AC04:** Continuous/Discrete Time
- **AC05:** Interconnection of Subsystems
- **AC06:** Controllability, Observability
- **AC07:** Inverse Systems
- **AC08:** Poles, Zeros
- **AC09:** Model Reduction
- **AC10:** Root Loci
- **AC11:** Control Diagrams

AD: Rational Matrix Analysis
- **AD01:** Equivalence Transformations
- **AD02:** Structural Indices
- **AD03:** Continuous/Discrete Time
- **AD04:** Interconnection of Subsystems
- **AD05:** Inverse Systems
- **AD06:** Poles, Zeros
- **AD07:** Model Reduction

AD08: Root Loci
AD09: Control Diagrams

AE: Frequency Response Analysis
AE01: Polar/Rectangular Coordinates
AE02: Interpolation
AE03: Inverse Systems
AE04: Continuous/Discrete Time
AE05: Interconnection of Subsystems

AF: Time Response Analysis
AF01: Scaling
AF02: Interpolation
AF03: Convolution, Deconvolution
AF04: Interconnection of Subsystems
AF05: Controllability, Observability
AF06: Change of Basis
AF07: Model Reduction

AG: Generalized State-space Analysis (including Periodic Systems)
AG01: Generalized Canonical and Quasi Canonical Forms
AG02: Change of Basis
AG03: Structural Indices
AG04: Continuous/Discrete Time
AG05: Interconnection of Subsystems
AG06: Controllability, Observability
AG07: Inverse and Dual Systems
AG08: Poles, Zeros, Gain
AG09: Model Reduction
AG10: Subspace Computation
AG11: Scalar and Multivariable Root Loci
AG12: Control Diagrams
AG13: Norms and Distances

Chapter - **S:** **Synthesis Routines**

SA: Auxiliary Routines

SB: State-space Synthesis (including Periodic Systems)
SB01: Eigenvalue/Eigenvector Assignment
SB02: Riccati Equations
SB03: Lyapunov Equations
SB04: Sylvester Equations
SB05: Minimum Variance Control
SB06: Dead Beat Control
SB07: Observers
SB08: Transfer Matrix Factorization
SB09: Realization Methods
SB10: Optimal Regulator Problems
SB11: Hierarchical Control

SB12: Decentralized Control
SB13: Non-interacting Control
SB14: Model Matching
SB15: Simulation

SC: Polynomial Matrix Synthesis
SC01: Eigenvalue/Eigenvector Assignment
SC02: Minimum Variance Control
SC03: Non-interacting Control
SC04: Model Matching
SC05: Parameter Optimization

SD: Rational Matrix Synthesis

SE: Frequency Response Synthesis

SF: Time Response Synthesis

SG: Generalized State-space Synthesis (including Periodic Systems)
SG01: Eigenvalue/Eigenvector Assignment
SG02: Generalized Riccati Equations
SG03: Generalized Lyapunov Equations
SG04: Generalized Sylvester Equations
SG05: Minimum Variance Control
SG06: Dead Beat Control
SG07: Observers
SG08: Transfer Matrix Factorization
SG09: Realization Methods
SG10: Generalized Optimal Regulator Problems
SG11: Hierarchical Control
SG12: Decentralized Control
SG13: Non-interacting Control
SG14: Model Matching
SG15: Simulation

Chapter - **D:** **Data Analysis**

DA: Auxiliary Routines

DB: Scaling, Interpolation

DC: Statistical Properties

DD: Trend Removal

DE: Covariances

DF: Spectra

DG: Discrete Fourier Transforms

DH: Z-Transforms

DJ: Prediction

DK: Windowing

DL: Filter Design

Chapter - **I:** **Identification**

IA: Auxiliary Routines

IB: Subspace Identification

IB01: Time Invariant State-space Systems
IB02: Generalized State-space Systems

IC: Parametric Methods
IC01: Covariance Methods
IC02: Deconvolution, Numerical Normal Equations
IC03: Bayes Estimation
IC04: Maximum Likelihood
IC05: Least Square Methods
IC06: Instrumental Variable Methods
IC07: Model Reference Methods
IC08: Prediction Error Methods
IC09: Stochastic Approximation
IC10: Order/Structure Determination

ID: General Methods
ID01: Parameter and State Estimation Combined
ID02: Use of Deterministic Signals
ID03: Evaluation of Input Signals
ID04: Test of Model Structure

IE: Non-parametric Methods
IE01: Frequency Analysis
IE02: Transient Analysis

Chapter - **F:** **Filtering**

FA: Auxiliary Routines

FB: Kalman Filters

FC: LPC Filters

FD: Fast Recursive Least Squares Filters

Chapter - **C:** **Adaptive Control**

CA: Auxiliary Routines

CB: Self Tuning Control
CB01: Minimum Variance Methods
CB02: Predictive Control Methods
CB03: Pole Placement Methods

CC: Model Reference Adaptive Control

CD: Parameter Estimation
CD01: Matrix Inversion Lemma
CD02: Square Root Algorithm
CD03: $U^T DU$ Transformation

Chapter - **N:** **Nonlinear Systems**

NA: Auxiliary Routines

NB: Volterra Series

NC: Bilinear Systems

ND: Describing Functions

NE: Stability Tests

Chapter - **B:** **Benchmark and Test Problems**

BA: Auxiliary Routines

BB: State-space Models
BB01: Optimal Control

BC: Rational Models

BD: Generalized State-space Models

BE: Polynomial Models

Appendix B

Subprogram Description Example

```
      SUBROUTINE AB01OD( STAGES, JOBU, JOBV, N, M, A, LDA, B, LDB, U,
     $                   LDU, V, LDV, NCONT, INDCON, KSTAIR, TOL, IWORK,
     $                   DWORK, LDWORK, INFO )
C
C     RELEASE 3.0, WGS COPYRIGHT 1997.
C
C     PURPOSE
C
C     To reduce the matrices A and B using (and optionally accumulating)
C     state-space and input-space transformations U and V respectively,
C     such that the pair of matrices
C
C        Ac = U' * A * U,    Bc = U' * B * V
C
C     are in upper "staircase" form. Specifically,
C
C             [ Acont     *    ]         [ Bcont ]
C        Ac = [                ],   Bc = [       ],
C             [   0    Auncont ]         [   0   ]
C
C        and
C
C                [ A11 A12  . . .  A1,p-1 A1p ]         [ B1 ]
C                [ A21 A22  . . .  A2,p-1 A2p ]         [ 0  ]
C                [  0  A32  . . .  A3,p-1 A3p ]         [ 0  ]
C        Acont = [  .   .   . . .    .     .  ],   Bc = [ .  ],
C                [  .   .     . .    .     .  ]         [ .  ]
C                [  .   .       .    .     .  ]         [ .  ]
C                [  0   0   . . .  Ap,p-1 App ]         [ 0  ]
C
C     where the blocks  B1, A21, ..., Ap,p-1  have full row ranks and
C     p is the controllability index of the pair.  The size of the
```

```
C     block Auncont is equal to the dimension of the uncontrollable
C     subspace of the pair (A, B).  The first stage of the reduction,
C     the "forward" stage, accomplishes the reduction to the orthogonal
C     canonical form (see SLICOT library routine AB01ND). The blocks
C     B1, A21, ..., Ap,p-1 are further reduced in a second, "backward"
C     stage to upper triangular form using RQ factorization. Each of
C     these stages is optional.
C
C     ARGUMENTS
C
C     Mode Parameters
C
C     STAGES  CHARACTER*1
C             Specifies the reduction stages to be performed as follows:
C             = 'F':  Perform the forward stage only;
C             = 'B':  Perform the backward stage only;
C             = 'A':  Perform both (all) stages.
C
C     JOBU    CHARACTER*1
C             Indicates whether the user wishes to accumulate in a
C             matrix U the state-space transformations as follows:
C             = 'N':  Do not form U;
C             = 'I':  U is internally initialized to the unit matrix (if
C                     STAGES <> 'B'), or updated (if STAGES = 'B'), and
C                     the orthogonal transformation matrix U is
C                     returned.
C
C     JOBV    CHARACTER*1
C             Indicates whether the user wishes to accumulate in a
C             matrix V the input-space transformations as follows:
C             = 'N':  Do not form V;
C             = 'I':  V is initialized to the unit matrix and the
C                     orthogonal transformation matrix V is returned.
C             JOBV is not referenced if STAGES = 'F'.
C
C     Input/Output Parameters
C
C     N       (input) INTEGER
C             The actual state dimension, i.e. the order of the
C             matrix A.  N >= 0.
C
C     M       (input) INTEGER
C             The actual input dimension.  M >= 0.
C
C     A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
C             On entry, the leading N-by-N part of this array must
C             contain the state transition matrix A to be transformed.
```

```
C             If STAGES = 'B', A should be in the orthogonal canonical
C             form, as returned by SLICOT library routine AB01ND.
C             On exit, the leading N-by-N part of this array contains
C             the transformed state transition matrix U' * A * U.
C             The leading NCONT-by-NCONT part contains the upper block
C             Hessenberg state matrix Acont in Ac, given by U' * A * U,
C             of a controllable realization for the original system.
C             The elements below the first block-subdiagonal are set to
C             zero.  If STAGES <> 'F', the subdiagonal blocks of A are
C             triangularized by RQ factorization, and the annihilated
C             elements are explicitly zeroed.
C
C     LDA     INTEGER
C             The leading dimension of array A.  LDA >= MAX(1,N).
C
C     B       (input/output) DOUBLE PRECISION array, dimension (LDB,M)
C             On entry, the leading N-by-M part of this array must
C             contain the input matrix B to be transformed.
C             If STAGES = 'B', B should be in the orthogonal canonical
C             form, as returned by SLICOT library routine AB01ND.
C             On exit with STAGES = 'F', the leading N-by-M part of
C             this array contains the transformed input matrix U' * B,
C             with all elements but the first block set to zero.
C             On exit with STAGES <> 'F', the leading N-by-M part of
C             this array contains the transformed input matrix
C             U' * B * V, with all elements but the first block set to
C             zero and the first block in upper triangular form.
C
C     LDB     INTEGER
C             The leading dimension of array B.  LDB >= MAX(1,N).
C
C     U       (input/output) DOUBLE PRECISION array, dimension (LDU,N)
C             If STAGES <> 'B' or JOBU = 'N', then U need not be set
C             on entry.
C             If STAGES = 'B' and JOBU = 'I', then, on entry, the
C             leading N-by-N part of this array must contain the
C             transformation matrix U that reduced the pair to the
C             orthogonal canonical form.
C             On exit, if JOBU = 'I', the leading N-by-N part of this
C             array contains the transformation matrix U that performed
C             the specified reduction.
C             If JOBU = 'N', the array U is not referenced and can be
C             supplied as a dummy array (i.e. set parameter LDU = 1 and
C             declare this array to be U(1,1) in the calling program).
C
C     LDU     INTEGER
C             The leading dimension of array U.
```

```
C             If JOBU = 'I', LDU >= MAX(1,N);  if JOBU = 'N', LDU >= 1.
C
C     V       (output) DOUBLE PRECISION array, dimension (LDV,M)
C             If JOBV = 'I', then the leading M-by-M part of this array
C             contains the transformation matrix V.
C             If STAGES = 'F', or JOBV = 'N', the array V is not
C             referenced and can be supplied as a dummy array (i.e. set
C             parameter  LDV = 1 and declare this array to be V(1,1) in
C             the calling program).
C
C     LDV     INTEGER
C             The leading dimension of array V.
C             If STAGES <> 'F' and JOBV = 'I', LDV >= MAX(1,M);
C             if STAGES = 'F' or JOBV = 'N', LDV >= 1.
C
C     NCONT   (input/output) INTEGER
C             The order of the controllable state-space representation.
C             NCONT is input only if STAGES = 'B'.
C
C     INDCON  (input/output) INTEGER
C             The number of stairs in the staircase form (also, the
C             controllability index of the controllable part of the
C             system representation).
C             INDCON is input only if STAGES = 'B'.
C
C     KSTAIR  (input/output) INTEGER array, dimension (N)
C             The leading INDCON elements of this array contain the
C             dimensions of the stairs, or, also, the orders of the
C             diagonal blocks of Acont.
C             KSTAIR is input if STAGES = 'B', and output otherwise.
C
C     Tolerances
C
C     TOL     DOUBLE PRECISION
C             The tolerance to be used in rank determination when
C             transforming (A, B). If the user sets TOL > 0, then
C             the given value of TOL is used as a lower bound for the
C             reciprocal condition number (see the description of the
C             argument RCOND in the SLICOT routine MB03OD);  a
C             (sub)matrix whose estimated condition number is less than
C             1/TOL is considered to be of full rank.  If the user sets
C             TOL <= 0, then an implicitly computed, default tolerance,
C             defined by  TOLDEF = N*N*EPS*MAX( NORM(A), NORM(B) ),  is
C             used instead, where EPS is the machine precision (see
C             LAPACK Library routine DLAMCH).
C             TOL is not referenced if STAGES = 'B'.
C
```

```
C     Workspace
C
C     IWORK   INTEGER array, dimension (M)
C             IWORK is not referenced if STAGES = 'B'.
C
C     DWORK   DOUBLE PRECISION array, dimension (LDWORK)
C             On exit, if INFO = 0, DWORK(1) returns the optimal value
C             of LDWORK.
C
C     LDWORK  INTEGER
C             The length of the array DWORK.
C             If STAGES <> 'B', LDWORK >= MAX(1, N*M + MAX(N,M) +
C                                                  MAX(N,3*M));
C             If STAGES =  'B', LDWORK >= MAX(1, M + MAX(N,M)).
C             For optimum performance LDWORK should be larger.
C
C     Error Indicator
C
C     INFO    INTEGER
C             = 0:  successful exit;
C             < 0:  if INFO = -i, the i-th argument had an illegal
C                   value.
C
C     METHOD
C
C     Staircase reduction of the pencil [B|sI - A] is used. Orthogonal
C     transformations U and V are constructed such that
C
C
C                        |B |sI-A      *  . . .  *      *       |
C                        | 1|    11       .      .      .       |
C                        |  |  A    sI-A    .    .      .       |
C                        |  |   21      22    . .       .       |
C                        |  |         .     .    *      *       |
C     [U'BV|sI - U'AU] = |0 |     0    .     .                  |
C                        |  |             A     sI-A     *      |
C                        |  |              p,p-1    pp          |
C                        |  |                                   |
C                        |0 |         0          0   sI-A       |
C                        |  |                            p+1,p+1|
C
C
C     where the i-th diagonal block of U'AU has dimension KSTAIR(i),
C     for i = 1,...,p. The value of p is returned in INDCON. The last
C     block contains the uncontrollable modes of the (A,B)-pair which
C     are also the generalized eigenvalues of the above pencil.
C
```

```
C     The complete reduction is performed in two stages. The first,
C     forward stage accomplishes the reduction to the orthogonal
C     canonical form. The second, backward stage consists in further
C     reduction to triangular form by applying left and right orthogonal
C     transformations.
C
C     REFERENCES
C
C     [1] Van Dooren, P.
C         The generalized eigenvalue problem in linear system theory.
C         IEEE Trans. Auto. Contr., AC-26, pp. 111-129, 1981.
C
C     [2] Miminis, G. and Paige, C.
C         An algorithm for pole assignment of time-invariant multi-input
C         linear systems.
C         Proc. 21st IEEE CDC, Orlando, Florida, 1, pp. 62-67, 1982.
C
C     NUMERICAL ASPECTS
C
C     The algorithm requires O((N + M) x N**2) operations and is
C     backward stable (see [1]).
C
C     CONTRIBUTOR
C
C     Release 3.0: V. Sima, Katholieke Univ. Leuven, Belgium, Nov. 1996.
C     Supersedes Release 2.0 routine AB01CD by M. Vanbegin, and
C     P. Van Dooren, Philips Research Laboratory, Brussels, Belgium.
C
C     REVISIONS
C
C     -
C
C     KEYWORDS
C
C     Controllability, generalized eigenvalue problem, orthogonal
C     transformation, staircase form.
C
C     ******************************************************************
C
C     .. Parameters ..
      DOUBLE PRECISION  ZERO, ONE
      PARAMETER         ( ZERO = 0.0D0, ONE = 1.0D0 )
C     .. Scalar Arguments ..
      CHARACTER*1       JOBU, JOBV, STAGES
      INTEGER           INFO, INDCON, LDA, LDB, LDU, LDV, LDWORK, M, N,
     $                  NCONT
      DOUBLE PRECISION  TOL
```

```
C     .. Array Arguments ..
      INTEGER           IWORK(*), KSTAIR(*)
      DOUBLE PRECISION  A(LDA,*), B(LDB,*), DWORK(*), U(LDU,*), V(LDV,*)
C     .. Local Scalars ..
      LOGICAL           LJOBUI, LJOBVI, LSTAGB, LSTGAB
      INTEGER           I0, IBSTEP, ITAU, J0, JINI, JWORK, MCRT, MM,
     $                  NCONT, NCRT
      DOUBLE PRECISION  WRKOPT
C     .. External Functions ..
      LOGICAL           LSAME
      EXTERNAL          LSAME
C     .. External Subroutines ..
      EXTERNAL          AB01ND, DGERQF, DLACPY, DLASET, DORGRQ, DORMRQ,
     $                  XERBLA
C     .. Intrinsic Functions ..
      INTRINSIC         MAX, MIN
C     .. Executable Statements ..
      ......
      ......
      ......
      ......
      ......
      ......
      RETURN
C *** Last line of AB01OD ***
      END
```

Appendix C

Subprogram Documentation Example

See next page.

AB01OD – SLICOT Library Routine Document

1 Purpose

To reduce the matrices A and B using (and optionally accumulating) state-space and input-space transformations U and V respectively, such that the pair of matrices

$$A_c = U^T A U, \qquad B_c = U^T B V$$

are in upper 'staircase' form. Specifically,

$$A_c = \begin{pmatrix} A_{cont} & * \\ 0 & A_{uncont} \end{pmatrix}, \qquad B_c = \begin{pmatrix} B_{cont} \\ 0 \end{pmatrix},$$

and

$$A_{cont} = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1,p-1} & A_{1p} \\ A_{21} & A_{22} & \cdots & A_{2,p-1} & A_{2p} \\ 0 & A_{32} & \cdots & A_{3,p-1} & A_{3p} \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & A_{p,p-1} & A_{pp} \end{pmatrix}, \qquad B_{cont} = \begin{pmatrix} B_1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

where the blocks B_1, A_{21}, ..., $A_{p,p-1}$ have full row ranks and p is the controllability index of the pair. The size of the block A_{uncont} is equal to the dimension of the uncontrollable subspace of the pair (A, B). The first stage of the reduction, the 'forward' stage, accomplishes the reduction to the orthogonal canonical form (see SLICOT library routine AB01ND). The blocks B_1, A_{21}, ..., $A_{p,p-1}$ are further reduced in a second, 'backward' stage to upper triangular form using RQ factorization. Each of these stages is optional.

2 Specification

```
      SUBROUTINE AB01OD( STAGES, JOBU, JOBV, N, M, A, LDA, B, LDB, U,
     $                   LDU, V, LDV, NCONT, INDCON, KSTAIR, TOL, IWORK,
     $                   DWORK, LDWORK, INFO )
C     .. Scalar Arguments ..
      CHARACTER*1       STAGES, JOBU, JOBV
      INTEGER           N, M, LDA, LDB, LDU, LDV, NCONT, INDCON, LDWORK,
     $                  INFO
      DOUBLE PRECISION  TOL
C     .. Array Arguments ..
      INTEGER           IWORK(*), KSTAIR(*)
      DOUBLE PRECISION  A(LDA,*), B(LDB,*), DWORK(*), U(LDU,*), V(LDV,*)
```

3 Arguments

3.1 Mode Parameters

STAGES CHARACTER*1
Specifies the reduction stages to be performed as follows:
= 'F': Perform the forward stage only;
= 'B': Perform the backward stage only;
= 'A': Perform both (all) stages.

JOBU CHARACTER*1
Indicates whether the user wishes to accumulate in a matrix U the state-space transformations as follows:
= 'N': Do not form U;
= 'I': U is internally initialized to the unit matrix (if STAGES $\neq$ 'B'), or updated (if STAGES = 'B'), and the orthogonal transformation matrix U is returned.

JOBV CHARACTER*1
Indicates whether the user wishes to accumulate in a matrix V the input-space transformations as follows:
= 'N': Do not form V;
= 'I': V is initialized to the unit matrix and the orthogonal transformation matrix V is returned.

JOBV is not referenced if STAGES = 'F'.

3.2 Input/Output Parameters

N (input) INTEGER
The actual state dimension, i.e. the order of the matrix A. N $\geq$ 0.

M (input) INTEGER
The actual input dimension. M $\geq$ 0.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the leading n-by-n part of this array must contain the state transition matrix A to be transformed. If STAGES = 'B', A should be in the orthogonal canonical form, as returned by SLICOT library routine AB01ND.
On exit, the leading n-by-n part of this array contains the transformed state transition matrix U^TAU. The leading n_{cont}-by-n_{cont} part contains the upper block Hessenberg state matrix A_{cont} in A_c, given by U^TAU, of a controllable realization for the original system. The elements below the first block-subdiagonal are set to zero. If STAGES $\neq$ 'F', the subdiagonal blocks of A are triangularized by RQ factorization, and the annihilated elements are explicitly zeroed.

LDA INTEGER
The leading dimension of array A. LDA $\geq$ max(1,N).

B (input/output) DOUBLE PRECISION array, dimension (LDB,M)
On entry, the leading n-by-m part of this array must contain the input matrix B to be transformed. If STAGES = 'B', B should be in the orthogonal canonical form, as returned by SLICOT library routine AB01ND.
On exit with STAGES = 'F', the leading n-by-m part of this array contains the transformed input matrix U^TB, with all elements but the first block set to zero. On exit with STAGES $\neq$ 'F', the leading n-by-m part of this array contains the transformed input matrix U^TBV, with all elements but the first block set to zero and the first block in upper triangular form.

LDB INTEGER
The leading dimension of array B. LDB $\geq$ max(1,N).

U (input/output) DOUBLE PRECISION array, dimension (LDU,N)
If STAGES $\neq$ 'B' or JOBU = 'N', then U need not be set on entry.
If STAGES = 'B' and JOBU = 'I', then, on entry, the leading n-by-n part of this array must contain the transformation matrix U that reduced the pair to the orthogonal canonical form.
On exit, if JOBU = 'I', the leading n-by-n part of this array contains the transformation matrix U that performed the specified reduction.
If JOBU = 'N', the array U is not referenced and can be supplied as a dummy array (i.e. set parameter LDU = 1 and declare this array to be U(1,1) in the calling program).

LDU INTEGER
The leading dimension of array U. If JOBU = 'I', LDU $\geq$ max(1,N); if JOBU = 'N', LDU $\geq$ 1.

V (output) DOUBLE PRECISION array, dimension (LDV,M)
If JOBV = 'I', then the leading m-by-m part of this array contains the transformation

matrix V.
If STAGES = 'F', or JOBV = 'N', the array V is not referenced and can be supplied as a dummy array (i.e. set parameter LDV = 1 and declare this array to be V(1,1) in the calling program).

LDV INTEGER
The leading dimension of array V. If STAGES $\neq$ 'F' and JOBV = 'I', LDV $\geq$ max(1,M); if STAGES = 'F' or JOBV = 'N', LDV $\geq$ 1.

NCONT (input/output) INTEGER
The order of the controllable state-space representation. NCONT is input only if STAGES = 'B'.

INDCON (input/output) INTEGER
The number of stairs in the staircase form (also, the controllability index of the controllable part of the system representation). INDCON is input only if STAGES = 'B'.

KSTAIR (input/output) INTEGER array, dimension (N)
The leading INDCON elements of this array contain the dimensions of the stairs, or, also, the orders of the diagonal blocks of A_{cont}. KSTAIR is input if STAGES = 'B', and output otherwise.

3.3 Tolerances

TOL DOUBLE PRECISION
The tolerance to be used in rank determination when transforming (A, B). If the user sets TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number (see the description of the argument RCOND in the SLICOT routine MB03OD); a (sub)matrix whose estimated condition number is less than 1/TOL is considered to be of full rank. If the user sets TOL $\leq$ 0, then an implicitly computed, default tolerance, defined by TOLDEF = n^2*EPS*max($\|A\|, \|B\|$), is used instead, where EPS is the machine precision (see LAPACK library routine DLAMCH). TOL is not referenced if STAGES = 'B'.

3.4 Workspace

IWORK INTEGER array, dimension (M)
IWORK is not referenced if STAGES = 'B'.

DWORK DOUBLE PRECISION array, dimension (LDWORK)
On exit, if INFO = 0, DWORK(1) returns the optimal value of LDWORK.

LDWORK INTEGER
The length of the array DWORK. If STAGES $\neq$ 'B', LDWORK $\geq$ max(1, N*M + max(N,M) + max(N,3*M)); if STAGES = 'B', LDWORK $\geq$ max(1, M + max(N,M)). For optimum performance LDWORK should be larger.

3.5 Error Indicator

INFO INTEGER
=0: successful exit;
<0: if INFO = $-i$, the i^{th} argument had an illegal value.

4 Method

Staircase reduction of the pencil $[B|sI-A]$. Orthogonal transformations U and V are constructed such that

$$[U^TBV|sI-U^TAU] = \left(\begin{array}{c|ccccc} B_1 & sI-A_{11} & * & \dots & * & * \\ 0 & A_{21} & sI-A_{22} & \ddots & \vdots & \vdots \\ \vdots & 0 & \ddots & \ddots & * & * \\ \vdots & \vdots & \ddots & A_{p,p-1} & sI-A_{pp} & * \\ 0 & 0 & \dots & 0 & 0 & sI-A_{p+1,p+1} \end{array}\right)$$

where the i-th diagonal block of U^TAU has dimension KSTAIR(i), for $i = 1,\dots,p$. The value of p is returned in INDCON. The last block contains the uncontrollable modes of the (A,B)-pair which are also the generalized eigenvalues of the above pencil.
The complete reduction is performed in two stages. The first, forward stage accomplishes the reduction to the orthogonal canonical form. The second, backward stage consists in further reduction to triangular form by applying left and right orthogonal transformations.

5 References

[1] Van Dooren, P.
The generalized eigenvalue problem in linear system theory.
IEEE Trans. Auto. Contr., AC-26, pp. 111-129, 1981.

[2] Miminis, G. and Paige, C.
An algorithm for pole assignment of time-invariant multi-input linear systems.
Proc. 21st IEEE CDC, Orlando, Florida, 1, pp. 62-67, 1982.

6 Numerical Aspects

The algorithm requires $O((n+m)n^2)$ operations and is backward stable (see [1]).

7 Further Comments

None.

8 Example

To reduce the matrices A and B to the upper 'staircase' forms U^TAU and U^TBV respectively, where

$$A = \begin{pmatrix} 17.0 & 24.0 & 1.0 & 8.0 & 15.0 \\ 23.0 & 5.0 & 7.0 & 14.0 & 16.0 \\ 4.0 & 6.0 & 13.0 & 20.0 & 22.0 \\ 10.0 & 12.0 & 19.0 & 21.0 & 3.0 \\ 11.0 & 18.0 & 25.0 & 2.0 & 9.0 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} -1.0 & -4.0 \\ 4.0 & 9.0 \\ -9.0 & -16.0 \\ 16.0 & 25.0 \\ -25.0 & -36.0 \end{pmatrix}.$$

8.1 Program Text

```
*     AB01OD EXAMPLE PROGRAM TEXT
*     RELEASE 3.0, WGS COPYRIGHT 1997.
*
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        ( NIN = 5, NOUT = 6 )
      INTEGER          NMAX, MMAX
      PARAMETER        ( NMAX = 20, MMAX = 20 )
      INTEGER          LDA, LDB, LDU, LDV
```

```
      PARAMETER        ( LDA = NMAX, LDB = NMAX, LDU = NMAX,
     $                   LDV = MMAX )
      INTEGER          LIWORK
      PARAMETER        ( LIWORK = MMAX )
      INTEGER          LDWORK
      PARAMETER        ( LDWORK = NMAX*MMAX + 4*MMAX )
*     .. Local Scalars ..
      DOUBLE PRECISION TOL
      INTEGER          I, INDCON, INFO, J, M, N, NCONT
      CHARACTER*1      JOBU, JOBV, STAGES
*     .. Local Arrays ..
      DOUBLE PRECISION A(LDA,NMAX), B(LDB,MMAX), DWORK(LDWORK),
     $                 U(LDU,NMAX), V(LDV,MMAX)
      INTEGER          IWORK(LIWORK), KSTAIR(NMAX)
*     .. External Subroutines ..
      EXTERNAL         AB01OD
*     .. Executable Statements ..
*
      WRITE ( NOUT, FMT = 99999 )
*     Skip the heading in the data file and read the data.
      READ ( NIN, FMT = '()' )
      READ ( NIN, FMT = * ) N, M, TOL, STAGES, JOBU, JOBV
      IF ( N.LE.0 .OR. N.GT.NMAX ) THEN
         WRITE ( NOUT, FMT = 99992 ) N
      ELSE
         READ ( NIN, FMT = * ) ( ( A(I,J), I = 1,N ), J = 1,N )
         IF ( M.LE.0 .OR. M.GT.MMAX ) THEN
            WRITE ( NOUT, FMT = 99991 ) M
         ELSE
            READ ( NIN, FMT = * ) ( ( B(I,J), J = 1,M ), I = 1,N )
*           Reduce the matrices A and B to upper "staircase" form.
            CALL AB01OD( STAGES, JOBU, JOBV, N, M, A, LDA, B, LDB, U,
     $                   LDU, V, LDV, NCONT, INDCON, KSTAIR, TOL, IWORK,
     $                   DWORK, LDWORK, INFO )
*
            IF ( INFO.NE.0 ) THEN
               WRITE ( NOUT, FMT = 99998 ) INFO
            ELSE
               WRITE ( NOUT, FMT = 99997 )
               DO 20 I = 1, N
                  WRITE ( NOUT, FMT = 99995 ) ( A(I,J), J = 1,N )
   20          CONTINUE
               WRITE ( NOUT, FMT = 99996 )
               DO 40 I = 1, N
                  WRITE ( NOUT, FMT = 99995 ) ( B(I,J), J = 1,M )
   40          CONTINUE
               WRITE ( NOUT, FMT = 99994 ) INDCON
               WRITE ( NOUT, FMT = 99993 ) ( KSTAIR(I), I = 1,INDCON )
            END IF
         END IF
      END IF
      STOP
*
99999 FORMAT (' AB01OD EXAMPLE PROGRAM RESULTS',/1X)
99998 FORMAT (' INFO on exit from AB01OD = ',I2)
99997 FORMAT (' The transformed state transition matrix is ')
```

```
99996 FORMAT (/' The transformed input matrix is ')
99995 FORMAT (20(1X,F8.4))
99994 FORMAT (/' The number of stairs in the staircase form = ',I3,/)
99993 FORMAT (' The dimensions of the stairs are ',/(20(I3,2X)))
99992 FORMAT (/' N is out of range.',/' N = ',I5)
99991 FORMAT (/' M is out of range.',/' M = ',I5)
      END
```

8.2 Program Data

```
 AB01OD EXAMPLE PROGRAM DATA
   5     2     0.0     F     N     N
   17.0   24.0    1.0    8.0   15.0
   23.0    5.0    7.0   14.0   16.0
    4.0    6.0   13.0   20.0   22.0
   10.0   12.0   19.0   21.0    3.0
   11.0   18.0   25.0    2.0    9.0
   -1.0   -4.0
    4.0    9.0
   -9.0  -16.0
   16.0   25.0
  -25.0  -36.0
```

8.3 Program Results

```
 AB01OD EXAMPLE PROGRAM RESULTS

 The transformed state transition matrix is
  12.8848   3.2345  11.8211   3.3758  -0.8982
   4.4741 -12.5544   5.3509   5.9403   1.4360
  14.4576   7.6855  23.1452  26.3872 -29.9557
   0.0000   1.4805  27.4668  22.6564  -0.0072
   0.0000   0.0000 -30.4822   0.6745  18.8680

 The transformed input matrix is
  31.1199  47.6865
   3.2480   0.0000
   0.0000   0.0000
   0.0000   0.0000
   0.0000   0.0000

 The number of stairs in the staircase form =   3

 The dimensions of the stairs are
  2    2    1
```

Appendix D

Keywords

A

adaptive control
adaptive estimation
adaptive filtering
adaptive observer
aggregate model
algebraic Riccati equation
approximate model
asymptotically stable
augmented Lagrangian
autocorrelation function
autoregressive model
autoregressive moving average model

B

balanced form
balancing
bang bang control
Bayes estimator
benchmarks
Bezoutian equation
bidiagonal form
bidiagonal matrix
bidiagonalization
bilinear control
bilinear transformation
block diagonal form
Bode diagram
boundary value problem
bounded control

C

D

discrete-time nonlinear system
discrete-time system
discretization
dissipative system
distances
distributed control
distributed parameter system
disturbance rejection
dual system

E

echelon form eigenfunction
eigenvalue
eigenvalue assignment
eigenvalue decomposition
eigenvector decomposition
equivalence transformation
estimation
explicit self tuning controller

F

factorization
fast Fourier transform
feedback control
feedforward control
Fourier analysis
frequency domain method
frequency response

G

gain control
generalized eigenvalue problem
generalized least-squares estimator
generalized state-space systems

H

Hamming window
Hankel matrix
Hann window
Hermite form

Markov process
matrix algebra
matrix exponential
matrix operations, elementary
matrix reordering
maximum likelihood estimation
maximum principle
minimal order
minimal realization
minimum variance controller
model inaccuracy
model matching
model reduction
model reference adaptive control
model validation
multivariable control systems
multivariable system

N

nonlinear estimation
nonlinear feedback system
nonlinear filtering
nonlinear regulator
nonlinear system
nonminimum phase system
norms
Nyquist criterion

O

observability
observer
observer Hessenberg form
on off control
optimal control
optimal control (finite horizon)
optimal filtering
optimal input design
optimal regulator
optimization
order determination
order test
orthogonal canonical form
orthogonal transformation
output feedback

P

Q

R

S

Schur form
self tuning controller
signal detection
similarity transformation
simulation
singular pertubation
singular pertubation approximation
singular subspace
singular value analysis
singular value decomposition
singular values
Smith predictor
spectral analysis
spectral factorization
square root covariance filtering
square root filtering
square root information filtering
stability
stability criteria
staircase form
state estimation
state feedback controller
state observers
state-space model
state-space representation
state-space transformation
steady state analysis
Stein equation
step response
stochastic control
stochastic differential equation
stochastic systems
structural decomposition
structural invariant
structure identification
subspace indentification
Sylvester equation
synthesis filter
system interconnection
system reduction
system response

Appendix E

Check List

A contribution to the library should contain at least the following items:

		Available	
1.	Location of the routine in the Library Index	◯ Yes	◯ No
2.	Short description of what the routine is about. If applicable, problems for which the routine is particularly suited, comparison with similar routines.	◯ Yes	◯ No
3.	Source text. The routine may have its original name. In any case the name need not satisfy the naming convention. This text should as far as possible meet the standards.	◯ Yes	◯ No
4.	Library Routine Document including an example program (text, data, results).	◯ Yes	◯ No
5.	Certification Program for validation and installation purposes plus data of relevant test problems.	◯ Yes	◯ No
6.	Keywords.	◯ Yes	◯ No
7.	References. This may be more than the references mentioned in the Library Routine Document.	◯ Yes	◯ No

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorenson, *LAPACK Users' Guide*, 2nd ed., Philadelphia, PA, 1995.

[2] M.J. Denham, C.J. Benson, *Implementation and Documentation Standards for the Software Library in Control Engineering (SLICE)*, Kingston Polytechnic, Control Systems Research Group, Internal report 81/3, November 1981.

[3] J.J. Dongarra, J. Du Croz, S. Hammarling, R.J. Hanson, *An Extended Set of Fortran Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 14, pp. 1–32, 1988.

[4] J.J. Dongarra, J. Du Croz, I. Duff, S. Hammarling, *A Set of Level 3 Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 16, pp. 1–28, 1990.

[5] J. Du Croz, *Draft Guidelines for Library Development*, NAG Central Office, Oxford.

[6] C.L. Lawson, R.J. Hanson, D.R. Kincaid, F.T. Krogh, *Basic Linear Algebra Subprograms for Fortran Usage*, ACM Transaction on Mathematical Software, pp. 308 - 323, 1979.

[7] Numerical Algorithms Group (NAG), *NAGWare f77 Tools Unix 2.0*, NP2419.

[8] Numerical Algorithms Group (NAG), *NAGWare Gateway Generator 2.0*, NP2770.

[9] Working Group on Software (WGS), *SLICOT Implementation and Documentation Standards (version 1.0)*, WGS-Report 90-1, Eindhoven University of Technology, May 1990.